Preprint
UCRL-JC-135151

# The Quest for a High Performance Boltzmann Transport Solver

*P.N. Brown, B. Chang, U.R. Hanebutte and C.S. Woodward*

This article was submitted to
6th International Conference on Applications of High-Performance Computing in Engineering
Maui, HI
January 26-28, 2000

**September 1, 1999**

# The Quest for a High Performance Boltzmann Transport Solver

P.N. Brown, B. Chang, U.R. Hanebutte & C.S. Woodward
*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Box 808, L-561, Livermore, CA 94551, E-mail: {pnbrown, chang1, ulf, cswoodward}@llnl.*

## Abstract

A 3-d Boltzmann transport code developed at Lawrence Livermore National Laboratory exploits concurrency (problem permitting) with respect to all phase space variables represented by direction, position and energy. To demonstrate the capabilities of the code, a highly resolved Boltzmann transport calculation with 27.6 billion unknowns (332 billion discretization points) was performed on 4992 processors of the IBM RS/6000 SP ASCI Blue machine at Lawrence Livermore National Laboratory. Detailed performance measurements were collected during this calculation. Analyzing the timing data revealed a relatively modest performance, i.e. obtained fraction of peak performance. A detailed study of the performance data confirmed that this modest performance was caused by low utilization of local processor performance, as well as low parallel efficiency of the intrinsically sequential sweep procedure. Both issues and their ramifications will be addressed in this paper.

## 1 Introduction

The ability to model the transport of neutral particles such as neutrons and photons through matter is of importance to many scientific and engineering activities. Such problems can often be modeled by the linear, steady-state Boltzmann Transport Equation (BTE), an integro-differential equation arising in deterministic models of neutral particle transport [7] given by

$$\Omega \cdot \vec{\nabla} \psi(\vec{r},\Omega,E) + \sigma(\vec{r},E)\psi(\vec{r},\Omega,E) =$$

$$\int_0^\infty \int_{S^2} \psi(\vec{r},\Omega',E')\sigma_s(\vec{r},E' \rightarrow E,\Omega \cdot \Omega')d\Omega' dE' + q(\vec{r},\Omega,E). \quad (1)$$

This equation models the flux $\psi$ of neutrons at position $\vec{r}$ and energy $E$ moving through a material in the direction given by the solid angle $\Omega$. The cross section $\sigma$ is a measure of how freely neutrons move through the material (and is related to the optical depth of the material), and the scattering cross section $\sigma_s$ is a measure of how many neutrons are scattered from energy $E'$ moving in direction $\Omega'$ to energy $E$ moving in direction $\Omega$. The term $q$ is a source term for neutrons at position $\vec{r}$, angle $\Omega$ and energy $E$.

We have developed a solution method for the BTE [1,3] and implemented it in a highly parallel code [2]. This code uses message-passing parallelism for spatial, energy, and angular decomposition. Thread-based parallelism can also be applied for the angular decomposition as well.

## 2 Solution Method

The discretization of phase space variables consists of a discrete ordinates ($S_n$) collocation in angle, Petrov-Galerkin finite-element method in space, and a multi-group approximation in energy [7]. In applying these discretization methods to (1), we need to approximate the scattering kernel $\sigma_s$. Since $\sigma_s$ only depends on $\mu$, the cosine of the angle between $\Omega$ and $\Omega'$, it is usually approximated by a truncated series of Legendre polynomials.

The integral over $\Omega'$ is approximated via a quadrature rule. The energy variable is discretized by piecewise constant finite elements, and the directional variable $\Omega$ are the points of the quadrature rule. Expressing the flux, $\psi$, in terms of a series of spherical harmonics leads to an integral equation in its coefficients of expansion (moments) of the form,

$$\phi = K\Gamma\phi + S, \quad (2)$$

where $\phi$ represents the moments of $\psi$, $\Gamma = \sigma_s / \sigma$, $K$ is an integral operator involving the inverse of the streaming operator,

$$\frac{1}{\sigma}\Omega \cdot \nabla + 1, \quad (3)$$

for each collocated direction $\Omega$, and $S$ depends on the source term $q$. A Biconjugate Gradient Stabilized (BiCGStab) iteration is used to solve for the $\phi$ values. The higher the order of the quadrature rule used in the discrete ordinates approximation, the more accurate the representation of the integral operator $K$ is, and the more costly it becomes to evaluate the $K\Gamma\phi$ term.

Inversion of the streaming operator (3) requires a sweep through the domain. For each direction, the sweep through the physical domain proceeds through the processor domain in a sequential manner, which can be quite expensive. In the next two subsections we describe our algorithms for handling this sweeping process, as well as how we apply parallel computation to solve (1).

## 2.1 3D Sweep

A sweep procedure through the three dimensional spatial domain is an essential part of the overall solution algorithm. Achieving good performance on a parallel computer is particularly difficult due to the fact that a single sweep is a sequential operation.

Domain decomposition is applied to the spatial grid resulting in a single subgrid which resides on a single processing node. Figure 1 depicts a single 3D wave front for one direction on a spatially distributed problem domain over 27 processing nodes. As shown, seven steps are required to propagate the front from one corner of the domain to its opposite corner. In [5,6], it was shown that the degree of concurrency can be increased by starting sweeps simultaneously (problem boundary conditions permitting) from all eight corners, i.e., all eight octants of the angular space. For 27 processing nodes this procedure is illustrated in Figure 2.

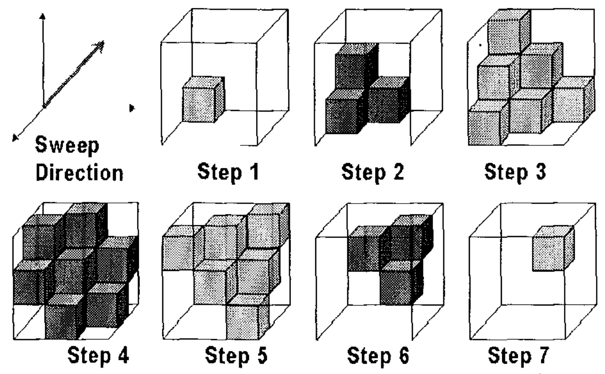

Figure 1: 3D wave front for one direction over 27 processor nodes.

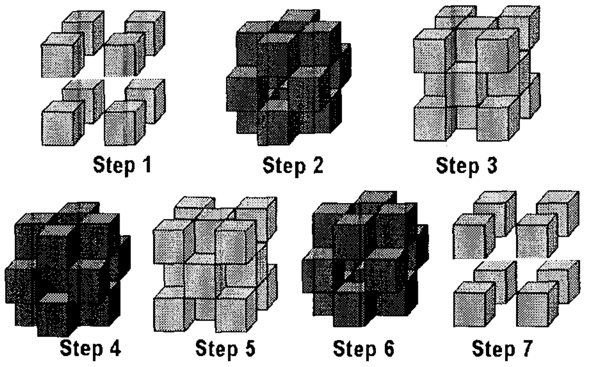

Figure 2: Simultaneous sweeps for all 8 octants on 27 processor nodes.

## 2.2 Parallelism

In the transport code we apply Cartesian decomposition for the spatial zones and assign spatial zones to nodes with a lexicographical ordering. A message-passing paradigm is used to communicate data between nodes. The energy dependence is decomposed into energy zones that can also be assigned to nodes lexicographically. However, if the physical problem has scattering from higher to lower energy groups, there will be a lower triangular dependence between the energy groups, and it is generally more appropriate to leave all groups on each node. When the groups are distributed, message passing is used for communication of data.

To utilize tightly coupled SMP-cluster architectures, where each node is comprised of a set of processors, we have implemented a multi-threaded paradigm for the angular decomposition [2]. Rather than enforcing a synchronize procedure which requires all processing nodes to synchronize at the end of each step, the code utilizes a data driven algorithm. Non-blocking send and receive operations provided by the MPI library are used to communicate data between neighboring processors during the sweep procedure. To utilize hyper-cluster architectures, where each processing node is comprised of a set of processors, the transport code takes advantage of the multi-thread paradigm.

Using Pthreads [9], each processing node computes multiple sweeps concurrently. Assuming n processors per node, the code starts n independent threads at the beginning of the 3D sweep procedure. Successively, each thread obtains required initial data for any sweep direction that has not yet been swept, which may require waiting. As soon as a thread receives the data, it starts to calculate that particular sweep. After completion of the single sweep data is sent to the neighbor processing nodes, and the thread is lined up to receive the next available work order.

Although the code allows for an MPI decomposition of the angular collocation points, this decomposition is not advisable for computations on SMP-clusters. Generally maximum efficiency is achieved if each processing node is responsible for all angular directions, due to task/data dependencies. Parallelism is best achieved here by utilizing threads.

## 3 Numerical Results

We have recently run what we believe is the largest physically relevant BTE calculation to date. This calculation simulated the neutron flux coming from a target chamber at LLNL shown in Figure 3. Over 900 separate surfaces were present in the physical problem specification. In this calculation, 14 orders of magnitude change in the physical properties and 4 orders of magnitude difference in the physical sizes needed to be modeled. The run was performed on the ASCI IBM RS/6000 SP machine at LLNL, a three-sector machine with each sector housing 432 four-way SMP nodes with either 1.5 or 2.5 gigabytes of memory for a total of 1,296 nodes. Each processor is a 332 MHz PowerPC 604e giving a system peak speed of 3.9 TeraFlops.

Our run used 1,248 nodes (logical 3D node layout: 8 x 12 x 13) of the machine with 4 processors on each node to give a total of 4,992 processors. The domain was decomposed into 150,508,800 total spatial zones (536 x 540 x 520) and 46 energy groups. A first order (in direction) approximation to the scattering kernel ($\sigma_s$) was used, which required solution for four unknown moments. A $S_6$ approximation was used to discretize the direction, which results in 48 quadrature points. This decomposition resulted in a total of 27,693,619,200 unknowns with 332,323,430,400 discretization points.

The run required 4 hours of compute time. The parallel decomposition consisted of 67 x 45 x 40 spatial zones per node. Down-scattering of energy between energy groups was present in the physical problem, so no parallel decomposition in energy was used. Pthreads were used to parallelize over the four processors in each node as described in Section 2.2 in order to achieve parallelism in the angular decomposition. Note that one thread per processor was used.

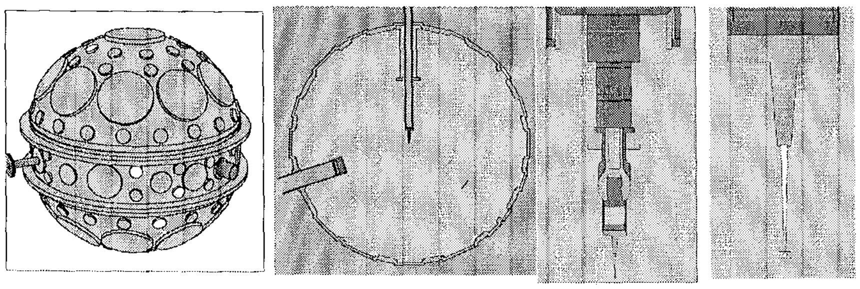

Figure 3: Scales for the laser target chamber: 240 in., 240 in., 8.4 in., and 0.24 in., respectively. Physical properties range over 14 orders of magnitude with values for a pure vacuum and various metals.

## 3. 1 Performance of the Threads

Figure 4 shows timings taken during the run of the local work for each processor for a single sweep. It is evident that although some directions took up to 1.5 seconds for a sweep, most took about 0.7 seconds, thereby indicating that the workload was evenly distributed over all threads used on all nodes. These results show that we have extracted a good bit of parallelism out of an operation that typically does not provide much parallelism at all.

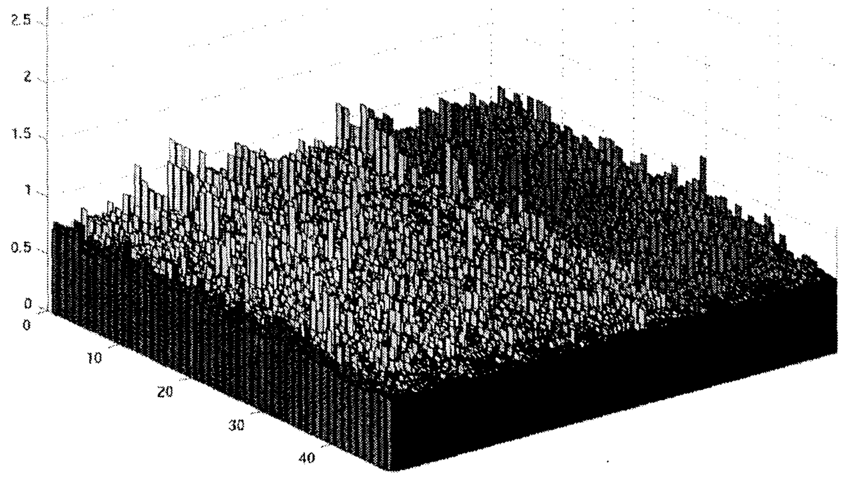

Figure 4: Bar graph showing times for computing local operations involved in a sweep for all directions on all processors. The left axis is the direction, and the right axis is the nodes.

In fact, if we assume that all directions took 0.7 seconds, then it would take 0.7 * 48 = 33.6 seconds to complete one sweep. If we then allow four parallel threads to work on the directions, it should take about 8.4 seconds to complete the numerical work for each sweep. However, in our run, we saw 45.14 seconds on average to complete a sweep, resulting in a parallel efficiency of 19%. We thus see that task/data dependency adds latent time to the parallel sweep algorithm. This effect can be seen in Figure 5, which shows the amount of busy time for each thread. It is readily evident that many threads spent much time waiting. The next section presents a strategy that might reduce this latency.

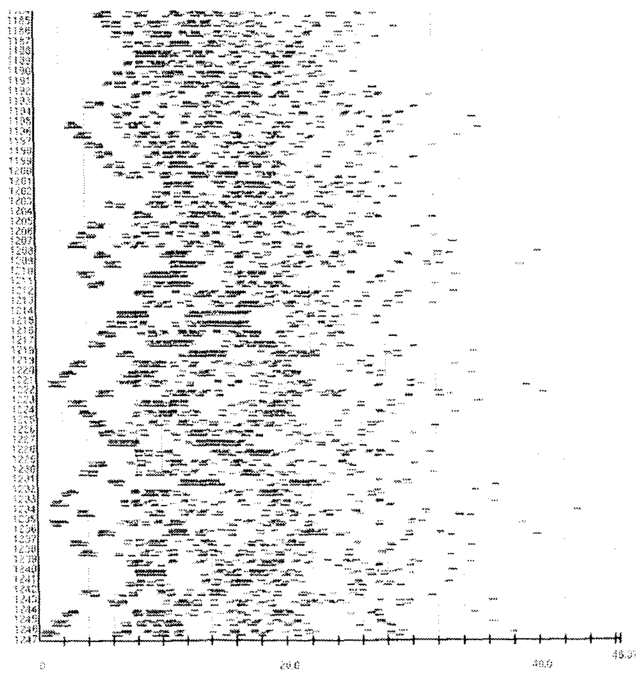

Figure 5: Observed time histogram of the sweep algorithm. Periods of work for each of the 4 threads of each node are indicated as a line segment. White space indicates idle time. Only a subset of the nodes is given here, however the time axis records the total time required for all processors to finish the sweep.

## 4 A new strategy for the parallel 3D sweep

In order to study the sweep algorithm and to investigate new strategies, a rudimentary simulation tool was developed which focuses on task/data dependency, while ignoring communication costs. The fact that communication costs are not considered is justified since the sweep algorithm overlaps communication and computations by utilizing non blocking send and receives. Table 1 summarizes simulator results for the sweep algorithm on 1248 nodes. The figure of merit is the parallel efficiency. Due to the simplifications of the simulator, the performance numbers can only be interpreted as upper bounds. For the computation discussed in this paper (i.e. 4 threads and a 1 on 1 mapping), the simulator predicts a 29.3% parallel efficiency, while measurements show 19% parallel efficiency. Two scenarios are presented in Table 1, one thread (i.e. processor) per node and four threads (i.e. processors) per node. Based on the assertion that the poor parallel performance of the sweep algorithm is due to spatial decomposition and the task/data dependency within the algorithm, one can increase the availability of tasks by dividing the domain into a number of subdomains which is greater than the number of. These subdomains are then randomly distributed among the nodes. While this strategy clearly violates data locality, it can provide a high degree of concurrency in an inherently sequential algorithm. The theoretical efficiency increased from 29.3% for 1248 domains to 50.3% for 9984 domains.

Table 1. Theoretical efficiencies obtained by the sweep simulator for 1248 nodes.

| | 1 thread | 4 threads |
|---|---|---|
| 1 on 1 mapping | 54.5% | 29.3% |
| 2 on 1 mapping | 51.9% | 35.8% |
| 4 on 1 mapping | 54.4% | 42.5% |
| 8 on 1 mapping | 58.7% | 50.3% |

Figures 6 and 7 depict the simulated time histograms for 1248 and 9984 domains mapped onto 1248 nodes (i.e. 1 or 8 domains per node) respectively. In this simulation 4 threads are active at each node. The time axis in Figure 6 and 7 is given in normalized units, which is based on the promise that each local domain sweep requires 1 unit of time. The time required for a single domain sweep is a linear function of domain size. Therefore, if one divides the domain into 8 times smaller domains the time per single domain reduces to 1/8th. That is, if one assumes that 41 seconds are needed for the sweep given in Figure 6, the total sweep time in Figure 7 would be 191/8 = 47.75 seconds. Relaxing the stipulation of a 1 on 1 mapping in the parallel code is not a trivial task. Data structures and a myriad of implementation details have to be changed. Therefore, our emphasis has shifted naturally to the single processor performance, as discussed in the next section.

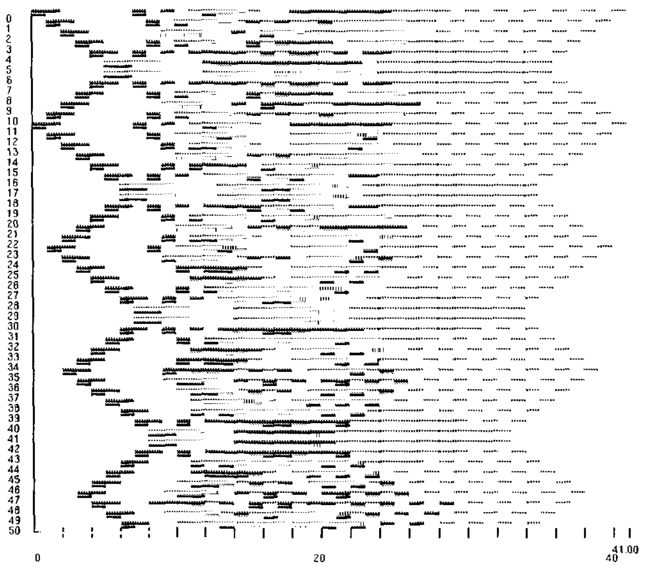

Figure 6: Result of a sweep algorithm simulation for 1248 domains mapped onto 1248 nodes (i.e. 1 domain per node). Each node contains a 4-way SMP for a total of 4992 processors. Only the first 51 nodes are shown.

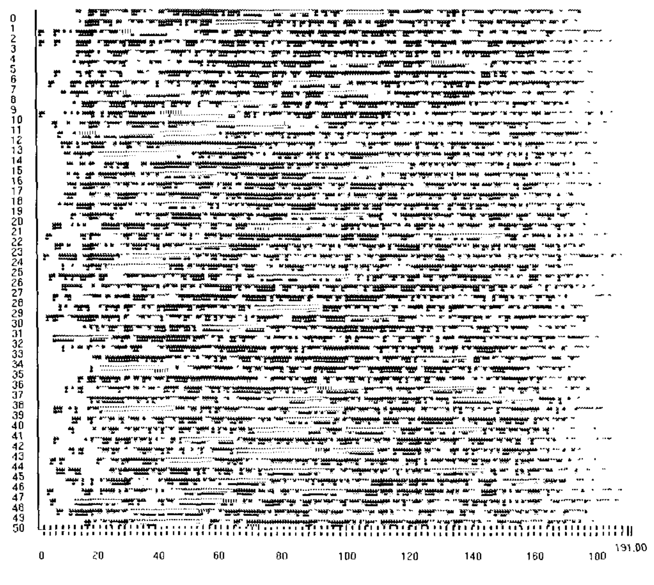

Figure 7: Result of a sweep algorithm simulation for 9984 domains mapped onto 1248 nodes (i.e. 8 domains per node). Each node contains a 4-way SMP for a total of 4992 processors. Only the first 51 nodes are shown.

## 5 Single processor performance and memory bandwidth

The peak performance of a single processor of the IBM ASCI Blue is 660 Mop/s, however in order to achieve peak, data has to be in the registers. If only a few operations per data item are performed, such as is the case for the BTE solver, memory bandwidth between main memory and the processing units becomes the limiting performance factor. In order to study the memory bandwidth bound for the four-way SMP node we modified the STREAM benchmark [8]. This benchmark is comprised of a set of operations, among which the triad operation closely resembles the most computationally intensive kernels of the BTE solver. For the triad operation, i.e. daxpy or saxpy, which requires 2 load, 2 floating point and 1 store operations, a surprisingly low number of 17.5 MFlop/s were measured when utilizing only one of the four processors on a SP node. An aggregated performance of 49.6 MFlop/s was obtained for the 4-way SMP on blocked data, while memory contention reduces the performance below the single processor limit if data is non-blocked. As stated above, the BTE kernels of the sweep algorithm are executed by each thread (i.e. each single processor) in a way similar to the modified triad benchmark. Taking the memory bandwidth bound into consideration, and aggressively restructuring the few main kernels of the BTE solver to take advantage of data reuse and loop unrolling, a speedup of 1.5 to 3 times could be achieved. Further details can be found in Ref. [4].

## 6 Conclusions

We have performed a very highly resolved neutron flux calculation on the ASCI IBM RS/6000 SP at Lawrence Livermore National Laboratory. This calculation involved solving the Boltzmann transport equation in phase space consisting of three spatial, two angular and one energy dimension. Analyzing the timing data of the successful transport calculation on 4992 processors revealed a relatively modest performance, i.e. actually obtained fraction of peak performance. A detailed study of the performance data confirmed that the modest performance was caused by low utilization of local processor performance (due to a demanding local memory access combined with a computer architecture with relative low memory bandwidth), as well as low parallel efficiency of the intrinsically sequential sweep procedure. Both issues have been addressed separately by utilizing a simulator to study different strategies for the sweep algorithm, and by extracting the dominant sequential kernels for further investigation and tuning. While a proposed modification to the sweep algorithm (requiring a major revision of the implementation) might provide a factor of 2 improvement, tuning and optimizing of the dominant sequential kernels can provide for an additional speedup of 1.5 to 3 times on the ASCI Blue.

## Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

## References

[1] Ashby, S.F., Brown, P.N., Dorr, M.R., & Hindmarsh, A.C. A Linear Algebraic Analysis of Diffusion Synthetic Acceleration for the Boltzmann Transport Equation *SIAM J. on Num. Anal.* 32:128-178, 1995.

[2] Brown, P.N., Chang, B., Dorr, M.R., Hanebutte, U.R., & Woodward, C.S. Performing Three-Dimensional Neutron Particle Transport Calculations on Tera Scale *High Performance Computing* '99 (part of the 1999 Advanced Simulation Technologies Conference), pp. 76-81, April 11-15, 1999, San Diego, CA.

[3] Brown, P.N., & Dorr, M.R. Spherical Harmonics Solutions of Neutron Transport Systems via Discrete *UCRL-JC-11976,* Lawrence Livermore National Laboratory, January 1995.

[4] Brunner, T.A., & Hanebutte, U.R. Investigation of Realistic Performance Limits for Tera-Scale, in preparation for submission to *High Performance Computing 2000* (part of the 2000 Advanced Simulation Technologies Conference), April 16-20, 2000, Washington, DC.

[5] Dorr, M.R. & Salo, E.M. Performance of a Neutron Transport Code with Full Phase Space Decomposition on the Cray Research T3D *Proceedings of the International Conference on Mathematics, and Computations, Reactor Physics, and Environmental Analysis,* American Nuclear Society, Portland, OR, April 30 – May 4, 1995, pp. 1535-1544.

[6] Dorr, M.R. & Still, C.H. Concurrent Source Iteration in the Solution of Three-dimensional, Multigroup, Discrete Ordinates Neutron Transport Equations, *Nucl. Sci. Eng* Vol. 122, No. 3, 1996.

[7] Lewis, E.E., & Miller Jr., W.F. *Computational Methods of Neutron Transport* Wiley, New York, 1984.

[8] McCalpin, J.D. STREAM: Sustainable memory bandwidth in high performance computers *Technical report* University of Virginia, 1995. http://www.cs.virginia.edu/stream

[9] Nichols, B., Buttlar, D., & Proulx Farrell, J. *Pthreads Programming* O'Reilly, 1996.